Second-Order Feed-Forward Rendering for Specular and Glossy Reflections

Lili Wang, Naiwen Xie, Wei Ke and Voicu Popescu

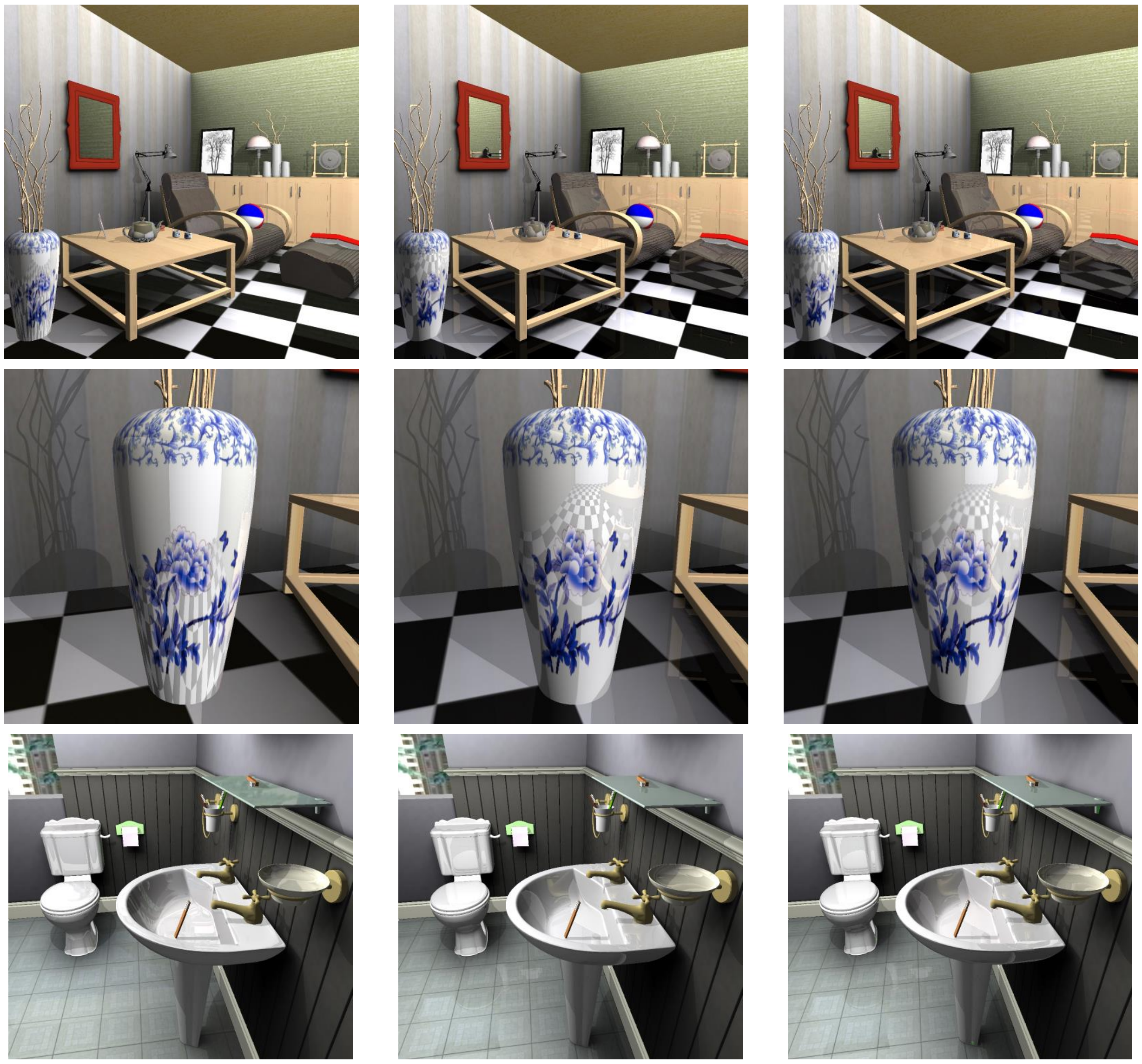

Figure 1. Specular reflections rendered with environment mapping (*left*), ray tracing (*middle*), and our method (*right*). Environment mapping produces incorrect reflections that fail to convey the reflected object's proximity to the reflector surface (e.g. table leg reflected in floor, floor reflected in vase). Our method renders reflections comparable to those rendered with ray tracing. The frame rate for our method is 15fps, compared to 1.1fps for ray tracing (i.e. Optix with BVH acceleration).

Abstract—The feed-forward pipeline based on projection followed by rasterization handles the rays that leave the eye efficiently: these first-order rays are modeled with a simple camera that projects geometry to screen. Second-order rays however, as for example those resulting from specular reflections, are challenging for the feed-forward approach. We propose an extension of the feed-forward pipeline to handle second-order rays resulting from specular and glossy reflections. The coherence of second-order rays is leveraged through clustering, the geometry reflected by a cluster is approximated with a depth image, and the color samples captured by the second-order rays of a cluster are computed by intersection with the depth image. We achieve quality specular and glossy reflections at interactive rates in fully dynamic scenes.

Index Terms— Specular reflections, glossy reflections, fully dynamic scenes, feed-forward rendering, interactive rendering

1 Introduction

Most interactive computer graphics applications render 3-D scenes in feed-forward fashion, by projection followed by rasterization. At a fundamental level, the approach is efficient since projection is a straight forward way to avoid considering ray/geometric-primitive pairs that do not produce an intersection. By comparison, the ray tracing pipeline requires acceleration schemes to avoid considering geometric primitives that do not intersect a given ray. However many scenes of interest contain specular reflective surfaces which extend and perturb first-order rays. The resulting higher order rays cannot be modeled with a simple camera that provides fast projection, and consequently the classic feed-forward pipeline cannot render reflections. The palliative approach for rendering reflections in interactive graphics applications is to approximate the reflected scene with a panoramic image, e.g. a cube map, and to look up the reflected rays into the cube map. This is a drastic approximation that produces large errors for reflected objects that are close to the reflector surface (Figure 1, left).

In this paper we propose extending the feed-forward pipeline to handle second-order rays. Our method is based on the fact that in the case of specular reflections second-order rays are locally coherent. We take advantage of this coherence by grouping second-order rays of nearby pixels into clusters. Although the rays in a cluster are coherent, they usually do not pass through a common point and thus they cannot be modeled with a conventional pinhole camera. One approach is to reduce the size of clusters until the pinhole approximation produces acceptable errors, but this is inefficient for complex reflectors that require small clusters and thus a large number of cameras.

Another approach is to model clusters of second-order rays using more powerful *non-pinhole* camera models, but such cameras introduce costly projection and non-linear rasterization, and even a small ray approximation error produces reflection discontinuity between clusters. *Instead of approximating the rays of the cluster, we approximate the geometry reflected by the cluster*. The reflected geometry is approximated by rendering a depth image for each cluster. The color samples captured by the second order rays of a cluster are computed by intersecting the rays with the cluster's depth image.

Our method produces quality reflections at interactive rates (see Figure 1 and accompanying video). No pre-computation is required, thus our method supports fully dynamic scenes. Second-order rays are clustered on the fly, directly in the output image, which brings support for general scenes, with large, complex, and numerous reflectors. The reflected scene geometry is approximated efficiently, on demand: our method only approximates the geometry needed for the reflections in the current frame, the approximation is done at the appropriate level of detail, and the approximation is done automatically, without a prerequisite partitioning of reflected geometry into objects. Our method also allows approximating glossy reflections (Figure 2) by intersecting the cluster depth image with multiple reflected rays per cluster pixel.

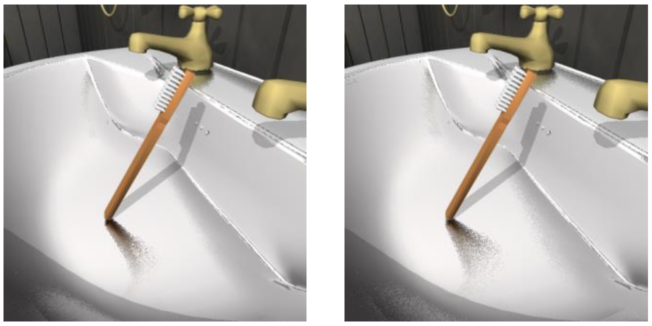
Figure 2. Glossy reflections rendered with our method, for two levels of glossiness, at 5 fps.

Our method essentially approximates second-order rays with one additional feed-forward rendering pass for each cluster. Although, in theory, our method could support higher-order rays, handling such rays is less efficient since they are less coherent, which translates into a larger number of clusters. Whereas incorrect first-order reflections are easily noticeable and disturbing, higher-order reflection inaccuracies are usually difficult to detect and as such they do not warrant the additional cost—we render higher order reflections using environment mapping.

The next section discusses prior work. Section 3 discusses our method in detail. Section 4 presents and discusses results. Section 5 concludes the paper and sketches directions for future work.

2 Related Work

The problem of rendering specular reflections at interactive rates has been approached from many directions.

Image based rendering & caching

One group of methods employs pre-computed or pre-acquired color samples. Such classic image-based rendering methods include the light field [20], the lumigraph [19], and view dependent texture mapping [29]. The lumigraph was modified from storing color samples to storing a ray to ray mapping, which allows changing the reflective and reflected object independently [21]. An outside-looking-in parameterization of the light field has been proposed by revolving a construction camera around the reflective object [22]. The environment light field map [18] goes in the opposite direction of achieving an inside-looking-out parameterization of the light field.

The idea of rendering acceleration by reusing pre-computed color has recently evolved into irradiance [37-39] and radiance [40-41] caching. Irradiance caching methods reuse shading results from nearby pixels and are efficient, but they only apply to diffuse surfaces. Radiance caching [40] overcomes the diffuse surface limitation by storing full incoming radiance, which is interpolated between pixels taking into account the pixels' BRDFs. Pre-convolved radiance caching [41] accelerates radiance caching by storing pre-computed shading expressions per surface as opposed to per pixel, at the cost of losing high-frequency detail.

These caching approaches are ill-suited for rendering specular reflections as the appearance of specular surfaces changes substantially even with a small change in viewpoint, which leads to impractically large ray databases. Moreover, dynamic scenes are challenging for image-based rendering methods since any change in the scene makes the ray database obsolete and re-computing the ray-database is too costly to be done per frame.

Ray tracing
The ray tracing pipeline [1, 2] naturally supports rendering reflections. The major concern is performance. The brute force approach of intersecting every scene triangle with every reflected ray is prohibitively expensive and the goal is to avoid performing intersection tests that do not yield an intersection. A multitude of acceleration schemes have been developed, including level of detail [5], KD-trees [3], bounding volume hierarchies (BVH) [4], and beam tracing [43], running on the CPU [11, 12] and on the GPU [13-15, 43]. Rendering specular and glossy reflections with ray tracing poses several challenges.

One challenge is the large number of per-pixel rays needed to achieve adequate reflection antialiasing. When the solid angle subtended by the reflected rays at a pixel is large, a large number of rays are needed for adequate sampling of reflected geometry. Another challenge is posed by glossy surfaces. Specular, mirror-like surfaces have coherent normals and therefore generate coherent reflected rays, which can be adequately sampled with a small number of per-pixel rays. Glossy surfaces however generate incoherent reflected rays and they require a large number of rays per pixel. A third important challenge is that, in the case of dynamic scenes, the data structure used to accelerate ray tracing has to be re-computed on the fly. For example, the Optix ray tracer used to render the comparison images in Figure 1 spends 910ms for the living room and 260ms for the bathroom per frame to re-construct its BVH tree.

As GPUs remain primarily feed-forward rendering machines, researchers have attempted to extend the feed-forward pipeline to rendering reflections. There are two fundamental options: processing the *reflected* triangles with the feed-forward pipeline, or processing the *reflective* triangles.

Feed-forward processing of reflected triangles
Consider a triangle that is first reflected before being projected onto the output image. Processing such a reflected triangle with the feed-forward pipeline requires overcoming two challenges. First, one has to be able to project onto the image plane a vertex that is first reflected. Second, one has to perform a non-linear rasterization of the reflected triangle (i.e. curved reflected triangle edges, non-linear variation of rasterization parameters within the triangle). The second challenge can be overcome by subdividing the reflected triangle until conventional, linear rasterization provides an acceptable approximation. However, the first challenge is difficult to overcome. If the reflector were a sphere, projecting a reflected vertex requires solving a quartic. For general reflectors modeled with a triangle mesh no closed-form projection exists.

The problem of projecting reflected vertices has been addressed in several ways. One method considers the reflected space subdivision induced by the reflector's triangles; a reflected vertex is projected by looking up the subdivision cell that contains it; the lookup is accelerated using an approximate representation called an explosion map [27]. Another method [8] leverages the coherence of reflected rays and approximates a group of reflected rays with a conventional planar pinhole camera. The planar pinhole cameras are stored at the leafs of a BSP tree that defines a *sample-based camera*. The sample-based camera projects reflected vertices with bounded error. A third method searches for the projection of a reflected vertex through a local search executed on the GPU [28]. All these methods scale poorly with reflector complexity and with the number of reflectors. Complex and numerous reflectors increase the complexity of the explosion map, of the sample-based camera, or of the search for the reflected vertex projection, and increase the number of projections for a given triangle due to multiple projections.

Feed-forward processing of reflective triangles
The other option for rendering reflections by projection followed by rasterization is to process the triangles that form the reflective surface. The vertices of the reflective triangle are projected as usual, vertex normals are interpolated over the projected triangle, and per-pixel reflected rays are computed straightforwardly. However, finding the color of the samples captured by the reflected rays is challenging. To avoid the complexity of ray tracing the scene in search of the reflected ray color, several methods resort to approximating the reflected scene geometry.

Environment mapping makes the drastic assumption that all reflected geometry is infinitely far away from the reflector [6, 7]. With this assumption, the reflected scene can be modeled with an environment map (typically parameterized as a cube map), and the reflected ray is simply looked up using solely its direction, ignoring the actual 3-D point from where it emanates. The reflection is antialiased through mipmapping in the environment map. The method scales well with reflector complexity and multiple reflections are handled at no extra cost—the fact that multiple reflected rays intersect the same region of the environment map has no consequence on performance. Due to its low cost and robustness, environment mapping is the method of choice for rendering reflections when performance is at a premium. However, when the reflected object is close to the reflective surface, environment mapped reflections are wrong, failing to convey the object's proximity to the reflective surface (Figure 1).

Environment maps have been extended to approximate single- and multiple-lobe BRDF glossy reflections [23, 24]. In an effort to improve reflected geometry approximation quality, environment maps have been enhanced with per-pixel depth [25]. However, the resulting environment map only captures surfaces visible from its center which leads to serious errors in the reflection due to missing samples as many reflected rays intersect surfaces that are not part of the environment map.

In order to reduce the missing sample errors one option is to subdivide the reflected scene into objects and to approximate each object individually for each reflector, using a billboard or a conventional depth image at first [26], and later using a non-pinhole depth image [9]. Good reflections are obtained, but subdividing the scene into reflected objects is not always possible and, when it is possible, the approach does not scale with scene complexity.

A second option is to approximate scene geometry with more powerful depth images. Layered depth images (LDIs) allow for a variable number of samples along a conventional camera ray [33] and they have been used to accelerate indirect illumination computation [34, 35]. Although LDIs avoid the redundancy of overlapping depth images, LDI construction is laborious (requiring depth peeling or merging overlapping depth images), which precludes their use in the context of dynamic scenes where LDIs would have to be constructed for every frame.

A third option is to use a flexible non-pinhole camera, such as the graph camera [36], to capture the entire scene in a single-layered image [9]. The graph camera offers closed-form projection and the graph camera depth image can be constructed (i.e. rendered) for every frame. However, graph camera constructors are limited to simple 2-D mazes with right angle turns, and therefore graph cameras cannot approximate well the reflected rays resulting from specular reflections in complex scenes like the ones considered in our paper (Figure 1).

Approximating reflected geometry has also been pursued by researchers aiming to accelerate ray tracing. Approximations include geometry fields that can be looked up to estimate the reflected ray color [16] and mip-mapped geometry images [17]. These approximations cannot be computed on the fly which precludes fully dynamic scenes with deforming objects.

Our method falls in this category of methods that feed-forward process reflective triangles and that approximate reflected geometry to simplify reflected ray / geometry intersection. Our method computes a quality approximation of the reflected scene for each frame which results in quality reflections for fully dynamic scenes, and it does not require partitioning the reflected scene into objects, which brings scalability with scene complexity.

3 Algorithm

Consider a scene that contains *diffuse* surfaces, i.e. surfaces with perfectly diffuse reflectance, *specular* surfaces, i.e. surfaces whose reflectance model is well approximated by a combination of a perfectly specular component and of a diffuse component, and *glossy* surfaces, i.e. surfaces whose reflectance model is well approximated by a combination of a single-lobe symmetrical BRDF and of a diffuse component. The scene is modeled with triangles.

3.1 Algorithm overview

Given a desired view V, the scene S is rendered from V with the following algorithm.

```
1. Render $S$ from $V$. For every pixel $p$ record:
    a. Diffuse component $p.rgb_d$
    b. Specular and glossiness levels $p.s$ and $p.g$
    c. Normal and depth $p.n$ and $p.z$
    d. Reflective object ID, $p.r_{ID}$
2. Cluster non-diffuse pixels
3. For every cluster $C$, finalize reflections as follows
    a. Construct cluster camera $K$
    b. Render $S$ with $K$ to obtain cluster depth image $D$
    c. For every pixel $p$ in $C$
        i. For every reflected ray $r_i$
            Intersect $r_i$ with $D$, i.e. $d_i = r_i \cap D$
        ii. Set non-diffuse component $p.rgb_n = G(d_i, p.g)$
        iii. Pixel color $p.rgb = LERP(p.rgb_d, p.rgb_n, p.s)$
```

The algorithm has three main steps. The **first step** takes a rendering pass over the scene to compute the diffuse component for every pixel, to set the pixel specular and glossiness levels, to compute the pixel normal and depth by conventional interpolation of vertex values, and to set the ID of the reflective object to which the pixel belongs. The specular level of a pixel $p.s$ ranges from 0 for perfectly diffuse to 1 for perfectly specular. The glossiness level of a pixel $p.g$ ranges from 0 for perfectly specular, mirror-like reflections, to 1 for a glossy surface with the widest BRDF lobe. The **second step** groups neighboring non-diffuse pixels with similar reflected rays into clusters as described in Section 3.2. The **third step** computes the reflections and finalizes the frame one cluster at the time. For each cluster, a planar pinhole camera is constructed to encompass all the reflected rays of the cluster as described in Section 3.3 (step 3.a above). Then the geometry reflected by the cluster is approximated by rendering the scene with the cluster camera (step 3.b).

The resulting depth image is used to finalize the computation of the color for the pixels in the cluster (step 3.c). For each pixel in the cluster, the depth image is intersected with reflected rays, as described in Section 3.4 (step 3.c.i above). For perfectly specular, mirror-like surfaces (i.e. a $p.g$ of 0), there is a single reflected ray per pixel, defined by the surface point, the pixel normal, and the eye position. The larger the glossiness factor $p.g$, the larger the number of reflected rays. The non-diffuse component of the current pixel is computed by blending the color samples d_i found at the reflected ray / depth image intersections based on the glossiness level $p.g$, using function $G(d_i, p.g)$, which corresponds to an application chosen BRDF. The final pixel color is computed by linearly interpolating the diffuse and non-diffuse pixel color components with weights defined by the pixels specular level $p.s$ (step 3.c.iii).

3.2 Non-diffuse pixel clustering

The first pass over the scene (step 1 in Section 3.1) computes the pixel normal which translates to a reflected ray. We take advantage of the coherence of per-pixel reflected rays by grouping nearby non-diffuse pixels into clusters. We have designed an algorithm for clustering non-diffuse pixels based on the following considerations:

a. There should be as few clusters as possible, since each cluster requires rendering the scene to construct its depth image.

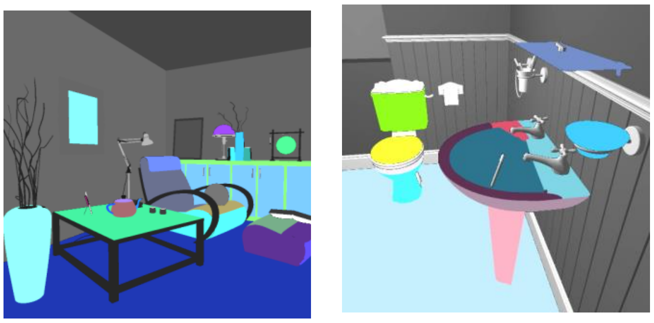
Figure 3. Visualization of reflective objects. Diffuse objects are shown in grey.

b. The cluster should be small enough such that a conventional planar pinhole camera constructed for the cluster captures the samples reflected by the cluster.
c. Clustering should be fast as it runs for each frame.

We assign non-diffuse pixels directly to their cluster by binning pixel normals into a 3-D array of bins whose resolution is adapted for each frame to achieve a good tradeoff between number of clusters and reflected ray modeling fidelity. Our algorithm proceeds as follows.

```
Offline
1. Partition non-diffuse triangles into reflective objects

Online, once per frame
2. For every reflective object $R$
      a. Set normalized screen area $a_R$
3. Compute the number of visible reflective objects, $n_v$
4. For each non-diffuse pixel $p$, assign $p$ to cluster $(i, j, k)$
      a. $i$ = MAXTHETABINS * $p.n.\theta / 360^o * a_R / n_v$
      b. $j$ = MAXPHIBINS * $p.n.\varphi / 180^o * a_R / n_v$
      c. $k = p.r_{ID}$
```

Binning normals based solely on their orientation can lead to grouping distant pixels in the same cluster. For example in Figure 1 (top row), the vase, the teapot, and the lamp have patches with identical normals. Grouping all three patches in the same cluster is inefficient since it would result in unnecessarily large depth images. This problem could be avoided by building contiguous clusters in bottom-up quadtree fashion in the output frame, but such an approach is slow.

Step 1. We group non-diffuse triangles offline into objects (**step 1**), and we prevent a cluster from spanning multiple reflective objects. This is done using the reflective object ID as a third dimension of the array in which normals are binned, in addition to the normal's spherical coordinates θ and φ. The partition of non-diffuse triangles follows the natural subdivision of the scene into objects. Figure 3 illustrates the 25 and 15 reflective objects for the living room and the bathroom scenes (Figure 1), respectively. We render reflections only for the non-diffuse pixels visible in the output frame—the offline partitioning of the scene into reflective objects is only used for fast clustering. Whereas the number of reflective objects is fixed, the resolution along the θ and φ dimensions of the array of bins is set online, for each reflective object and for each frame. The θ and φ dimensions of the bins depend on two

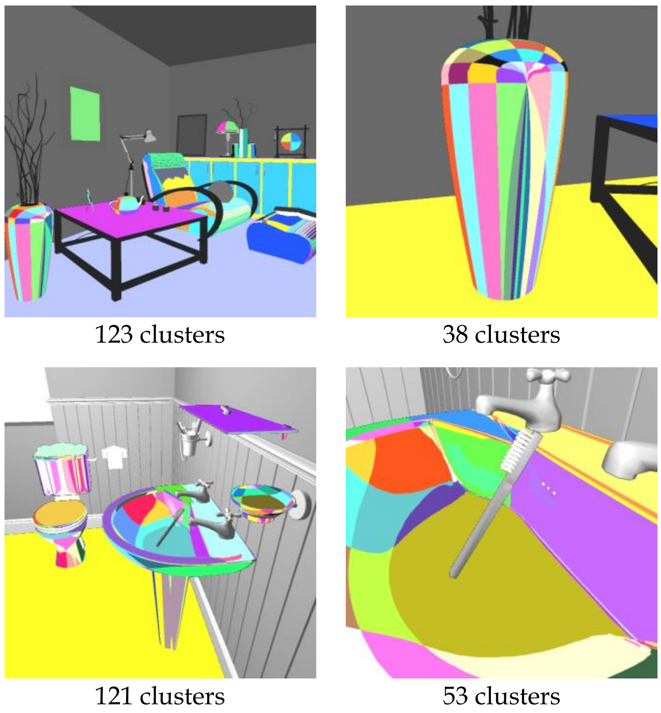

Figure 4. Visualization of pixel clusters used to render the reflections in Figs 1 and 2. Diffuse pixels are shown in grey.

quantities.

Step 2. One quantity, a_R, measures the footprint of the reflective object in the output frame, as the percentage of output frame pixels where the reflective object is visible.

Step 3. The second quantity is the number of reflective objects n_v that are visible in the output frame, computed as the number of reflective objects whose a_R is not 0 (**step 3**).

Step 4. Each pixel is assigned to a cluster based on its normal and on the index of the reflective object to which it belongs. The maximum possible θ and φ resolution MAXTHETABINS x MAXPHIBINS is modulated using a_R and n_v. MAXTHETABINS and MAXPHIBINS are constants that we set to 13 and 8, respectively, for all examples shown in the paper. The larger the relative footprint of the object, the finer the bins, and the larger the number of reflective objects, the coarser the bins. Although the maximum number of bins for the two scenes is 25 x 13 x 8 = 2,600 and 15 x 13 x 8 = 1,560, respectively, the number of clusters is given by the number of bins that are not empty. For example, in the case of a single large sphere that covers the entire screen, the maximum number of clusters is 1 x 13 x 8 / 2= 52, which accounts for the fact that only half of the sphere is visible.

Figure 4 illustrates the clusters used to render the reflections in the four images from Figures 1 and 2. The floor defines a single cluster. The table top and the floor define two different clusters, although they are oriented the same way. The vase defines more clusters when seen in more detail (top right versus top left). The spherical coordinate system used over-samples clusters near the pole (see top right image in Figure 4), a small disadvantage outweighed by the advantage of its simplicity. Clustering based on normals as opposed to based on re-

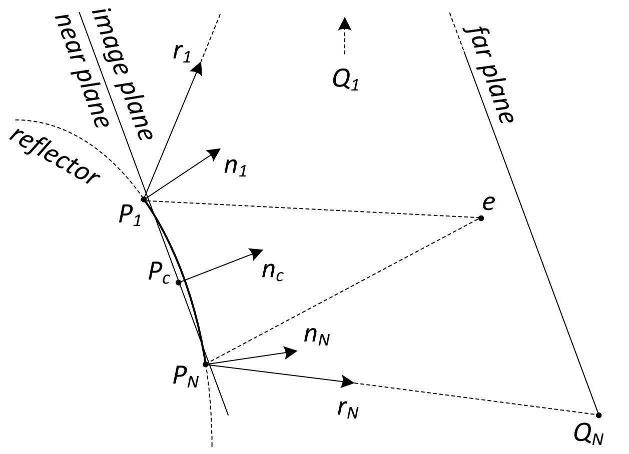

Figure 5. Construction of the image plane (also the near plane) and of the far plane for the cluster camera.

flected rays avoids over-clustering at the silhouette of reflective objects and has good frame to frame stability.

3.3 Cluster camera construction

Once clustering is complete, reflections are computed for one cluster at the time. The first step is to construct a conventional planar pinhole camera for the current cluster that allows approximating the geometry reflected by the cluster. We use a conventional camera for three fundamental reasons. First, conventional camera allows rendering the scene geometry efficiently to obtain a depth image that approximates the reflected geometry. Second, reflected rays project to straight lines onto depth images constructed with conventional cameras, which makes ray / depth image intersection efficient. Third, a conventional camera models the reflected rays of a planar cluster perfectly. Planar specular surfaces abound in man-made scene and handling them well is an important reflection rendering algorithm design consideration.

The goal is for the camera to capture all the samples reflected by the cluster. One consideration is for the camera to have enough field of view, such that the camera frustum contains all the reflected rays of the cluster. A second consideration is for the rays of the camera to approximate the reflected rays of the cluster as closely as possible in order to avoid disocclusion errors in the reflection. A disocclusion error occurs when a reflected ray intersects a surface at a sample that is missing from the depth image rendered with the cluster camera. The camera cluster is constructed as follows (Figure 5).

```
1. Set image plane through 3-D point $P_c$ and normal $n_c$
    a. $P_c = \sum P_i / N$
    b. $n_c = \sum n_i / N$
2. Set far plane at distance $D$
3. Set center of projection $e_c$
    a. Compute $e_0$ = Reflect($e$, $P_c$, $n_c$)
    b. Displace $e_0$ along $e_0P_c$: $e_c = e_0 + (P_c - e_0)f$
4. Set the image frame $aabb$
5. Set the image resolution $w \times h$
```

Step 1 The image plane of the cluster camera is defined by the cluster centroid and the cluster normal. The cluster centroid is the average of the 3-D reflector surface points over all cluster pixels (N is the number of pixels in the cluster). The cluster normal is the average over all cluster pixel normals. The image plane also serves as near plane. Figure 5 illustrates cluster camera construction in 2-D, for clarity. A curved reflector is partitioned into a cluster between points P_1 and P_N, where the surface normals are n_1 and n_N and the reflected rays are r_1 and r_N. The output frame center of projection is e.

Step 2 For specular reflections, the far plane of the cluster camera is set to be parallel to the image plane at a distance D equal to the scene diameter for specular reflections (Figure 5). For glossy reflections the far plane is set closer to the image plane, as described in Section 3.5.

Step 3 The center of projection e_c of the cluster camera is defined such that the cluster camera rays approximate the reflected rays of the cluster as well as possible. We construct e_c such that the axis aligned bounding box $aabb_n$ of the projections of the *near* reflected ray endpoints be of similar size to the axis aligned bounding box $aabb_f$ of the projections of the *far* reflected ray endpoints. In Figure 5, the near endpoints of reflected rays r_1 and r_N are P_1 and P_N; the far endpoint for ray r_N is Q_N, and for r_1 it is Q_1 (actual location of Q_1 is not shown to keep the figure compact).

We set e_c in two steps. First, the output frame center of projection e is reflected over the cluster camera image plane to e_0 (Figure 6). Then, e_c is computed by displacing e_0 towards or away from the centroid of the cluster P_c. The displacement is controlled by a scalar value f, which is set such that the diagonal of the 2-D AABB $aabb_f$ be approximately equal to the diagonal of the 2-D AABB $aabb_n$. We set f as shown in Equation 1, where d_n is the length of the diagonal of $aabb_n$ when e_c is at e_0, d_f is the length of the diagonal of $aabb_f$ when e_c is at e_0, and d_F is the length of the axis aligned bounding box of the far endpoints of the reflected rays on the far plane. As e_c moves on e_0P_c, the projections P_i' of points P_i do not change much, as points P_i are close to the image plane. Consequently, the length of the diagonal of $aabb_n$ is approximately constant, and Equation 1 approximates it to the length d_n it has when e_c = e_0. On the other hand, the length of the diagonal of $aabb_f$

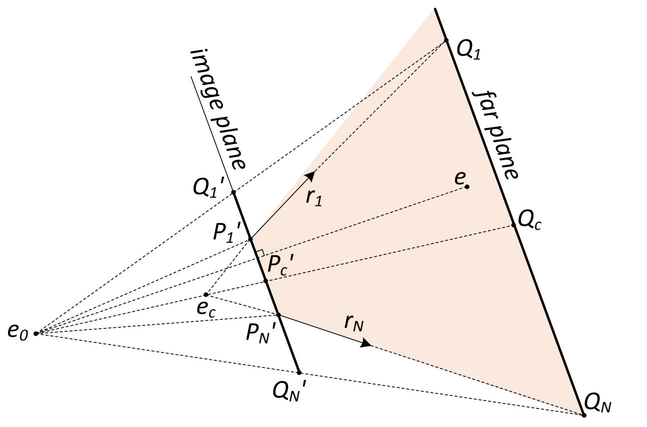

Figure 6. Construction of the eye e_0 of the cluster camera.

is sensitive to the position of e_c on e_0P_c. Equation 1 sets e_c such that the length of the diagonal of $aabb_f$ is equal to d_n.

$$f = \frac{d_n(d_F - d_f)}{d_f(d_F - d_n)} \qquad (1)$$

Figure 6 illustrates the case of the convex reflector from Figure 5. Points Q_1', P_1', P_c', P_N', Q_N' correspond to the cluster camera image plane projection from e_0 of the corresponding points from Figure 5. The length of segment $Q_1'Q_N'$ corresponds to d_f, that of $P_1'P_N'$ corresponds to d_n, and that of Q_1Q_N corresponds to d_F. At e_0 $aabb_n$ is smaller than $aabb_f$, i.e. $d_f > d_n$. Since $d_F > d_f$ and $d_F > d_n$, $0 < f < 1$. The displacement moves e_c from e_0 towards P_C. This decreases $aabb_f$ with respect to $aabb_n$. The cluster camera with center of projection e_c approximates the reflected rays better than a cluster camera with center of projection at e_0. The direction of e_cP_1' is closer to the direction of r_1 than is the direction of e_0Q_1'.

For a cluster corresponding to a planar reflector, all reflected rays intersect at e_0, and consequently the cluster camera should use e_0 as a center of projection. Such a cluster camera models the reflected rays perfectly and there are no disocclusion errors. For a planar reflector the near and far endpoints project from e_0 to the same image plane point. Consequently $aabb_n$ and $aabb_f$ are identical, $d_n = d_f$, and, according to Equation 1, f becomes 1, which implies that e_c is set to e_0 as desired.

Step 4 The image frame is set at by projecting the near and far reflected ray endpoints with the finalized center of projection e_c. The image frame is the axis aligned bounding box of these projections. This way the cluster camera has a field of view that is guaranteed to encompass all the reflected rays of the cluster (orange shaded area in Figure 6).

Step 5 The image resolution is set to match the resolution of the cluster. The 3-D points of a few pairs of cluster pixels that are either in consecutive rows or in consecutive columns in the output image are projected with the cluster camera. The average distance between the pairs of projections are used to define the pixel size p. The cluster camera resolution is defined as w x h, where $w = aabb.w / p$, and $h = aabb.h \ / \ p$.

3.4 Reflected ray / depth image intersection

Once the camera cluster is complete, the scene is rendered with it to obtain the cluster depth image. A depth image is a powerful approximation of geometry: the approximation can be constructed quickly through conventional rendering to obtain a frame buffer with color and depth per pixel, the depth image captures geometry with controllable level of detail, and one can intersect a depth image with a single ray efficiently. The efficient intersection between a ray and a depth image is well known—it has been used in inverse image-based rendering by 3-D warping [31], in rendering surface geometric detail [32], and in rendering reflections [26]. We briefly sketch the algorithm here for completeness.

Given a ray r and a cluster depth image DI, the closest .intersection between r and DI, if any, is found by projecting r onto DI. Let r' be the projection of r with the cluster camera that rendered DI; r' is traversed from the near endpoint to the far endpoint with one-pixel steps. Let a and b be the previous and current steps on r'. If the 2-D segments [(0, z_{ra}), (1, z_{rb})] and [(0, $DI[a]$), (1, $DI[b]$)] intersect, and intersection is found, and the search stops. z_{ra} is the depth along the ray at a, and $DI[a]$ is the depth in the depth image at a. A measure of depth that is linear in screen space is used, i.e. proportional to $1/z$. If the end of r' is reached there is no intersection.

3.5 Glossy reflections

A point on a glossy surface does not reflect along a single direction, but rather along a solid angle centered at the specularly reflected ray (i.e. the ray obtained by reflecting the output image ray over the point's normal). We support glossy reflections with the following three modifications to the algorithms described above for specular reflections.

(1) The field of view of the cluster camera has to be constructed to take into account the non-zero solid angle subtended by a glossy reflected cone. We do this by extending the field of view computed at Step 4 of the cluster camera construction algorithm (Section 3.3) with the angle of the reflected cone. This way the resulting cluster depth image captures the additional geometry reflected by the glossy cluster.

(2) Glossy surfaces only have well defined reflections close to the reflector surface. We take advantage of this fact by setting the far plane of a cluster camera constructed to render a glossy reflection based on the glossiness level. For surfaces that are more matte, the far plane can be closer to the near plane, compared to surfaces that are more specular. Bringing in the far plane as much as possible reduces the amount of geometry that has to be reflected. When no object is sufficiently close to a glossy surface, the resulting cluster depth image is empty, and no subsequent ray / depth image intersections are needed. To provide for a gradual fade away of a glossy reflection as a reflected surface moves progressively farther from the glossy surface, the glossy reflection is blended with the diffuse color of the surface with a weight that decreases to 0 as the distance to the reflected surface becomes the distance to the far plane.

(3) Glossy reflections require multiple reflected rays per reflector surface point. We intersect several reflected rays with the cluster depth image, for each glossy surface pixel. The reflected rays sample uniformly a cone with apex at the surface point and with an axis defined by the reflected ray generated by the pixel normal. The solid angle covered by the cone depends on the glossiness level. For surfaces close to specular, the angle is small, and therefore the number of rays is small. For a more matte surface the angle is larger requiring additional rays. For the examples shown in this paper the number of rays per glossy pixel ranges from 8 to 49.

4 Results and Discussion

We tested our method on two indoor scenes with numerous specular and glossy reflections. The living room scene

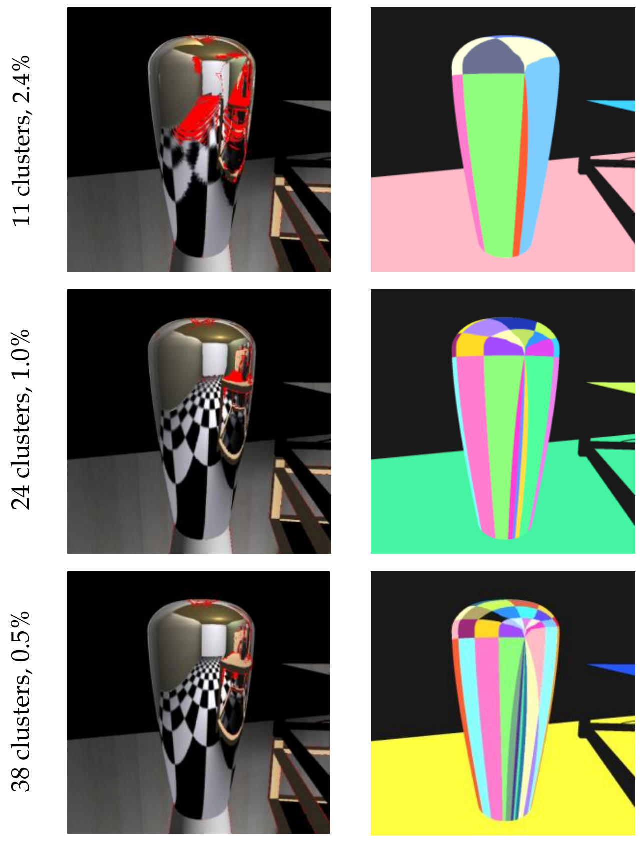

Figure 7. Disocclusion errors (red) decreasing as more clusters are used, for the image in the middle row of Figure 1. Diffuse pixels are shown in black and only the reflective component is shown for non-diffuse pixels. The bottom row shows that there are no reflection discontinuities between adjacent clusters.

(Figure 1, top and middle rows) has 286K triangles, out of which 156K are diffuse and 132K are non-diffuse. The bathroom scene (Figure 1, bottom row) has 90K triangles, out of which 44K are diffuse and 46K are non-diffuse.

4.1 Quality

Specular (mirror like) reflections

As shown in Figure 1 and in the accompanying video, our method produces quality specular reflections. Unlike in the case of environment mapping, objects close to the reflector are reflected correctly, conveying the proximity between the reflected and reflecting object. There is no reflection discontinuity between clusters because the clusters have a slight overlap, which prevents any gaps, and because the reflected rays are continuous over the smoothly changing reflector surface, which prevents any misalignment of the reflection from cluster to cluster (see Figure 7, bottom).

The reflections rendered with our method are comparable to reflections rendered by ray tracing. Throughout this paper and the video, reflections rendered with our method are rendered at a resolution of 512 x 512 with uniform 2x2 super-sampling (i.e. 1,024 x 1,024 before output frame reconstruction); the reflections rendered by ray tracing use an equivalent 512x512 output image resolution with 4 rays per pixel. We use NVIDIA's Optix ray tracer [30]. Figure 8 shows pixel value differences between our method and ray tracing. The average absolute pixel channel differences (i.e, L1 norm) are small, i.e. 6, 3, and 9, for each of the three rows, respectively (we use eight bit RGB channels with values from 0 to 255).

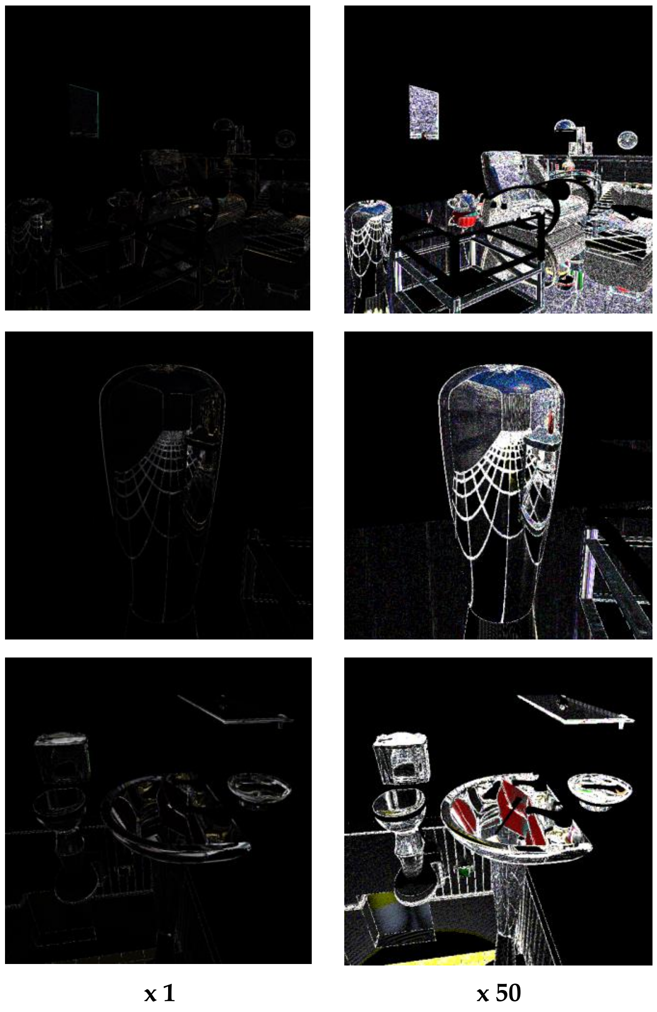

Figure 8. (*Left*) Difference images between our method and ray tracing for the three rows in Figure 1. (*Right*) same difference images with intensities scaled up by a factor of 50.

One reason for the difference is the large angle between reflected rays at the reflector edges, which leads to minification errors. Another reason for the difference is the slightly different sampling of the diffuse objects, which produces differences most visible at color edges. Our method uses bilinear interpolation of the intermediate sampling provided by the depth image, whereas ray tracing samples the diffuse geometry directly with additional rays.

Differences are also caused by surfaces that should be visible in the reflection at a cluster but that are not captured by the cluster depth image. The reflected rays do not pass exactly through the cluster camera's center of projection so it can happen that a few reflected rays reach surfaces that are not visible to the cluster's camera. We measured the number of pixels per frame where such disocclusion errors occur over a sequence of 1,000 frames.

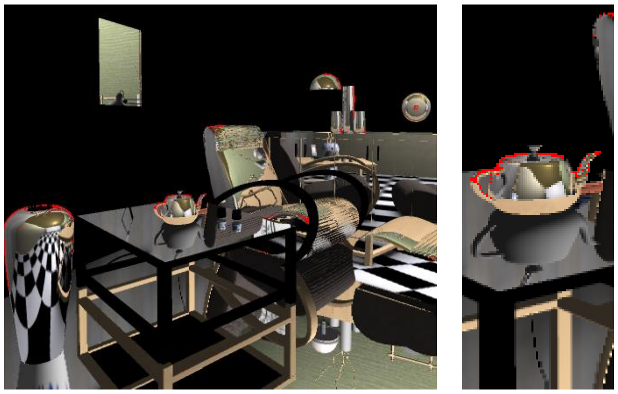
Figure 9. Visualization in red of pixels whose reflected rays are looked up in an environment map, for Figure 1 (top). Diffuse pixels are shown in black and only the reflective component is shown for non-diffuse pixels.

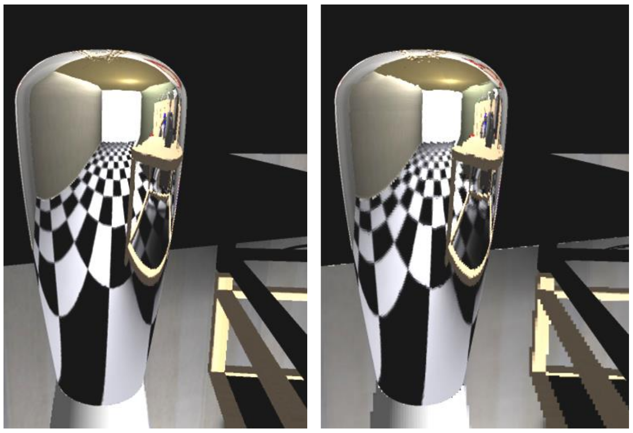
Figure 10. Reflections rendered with our algorithm (*left*) and with depth images with 3x3 lower resolution than the resolution computed by our algorithm (*right*). Diffuse pixels are shown in black and only the reflective component is shown for non-diffuse pixels.

The maximum / average percentage of disocclusion error pixels is 1.3% / 0.62% for the living room scene, and 3.5% / 2.75% for the bathroom scene. The disocclusion error is controlled by reducing the size of the clusters. The fewer the reflected rays that are approximated with a single planar pinhole camera, the higher the approximation fidelity (Figure 7).

Finally, when curved reflectors have a small screen footprint, the cluster has only one or a few rays, which can lead to not finding an intersection between the reflected ray and the depth image of the cluster. When an intersection is not found, the reflected ray is looked up in an environment map (Figure 9).

Our method approximates the reflected geometry for each output frame using depth images, which allows adapting the level of detail of the reflected geometry as needed for the current frame. The resolution of the depth image rendered for a cluster is commensurate with the resolution of the cluster of reflected pixels. Figure 10 shows that a lower resolution for the depth images would lead to blurriness (distant part of floor reflected in vase) and jagged edges (table leg reflected in floor).

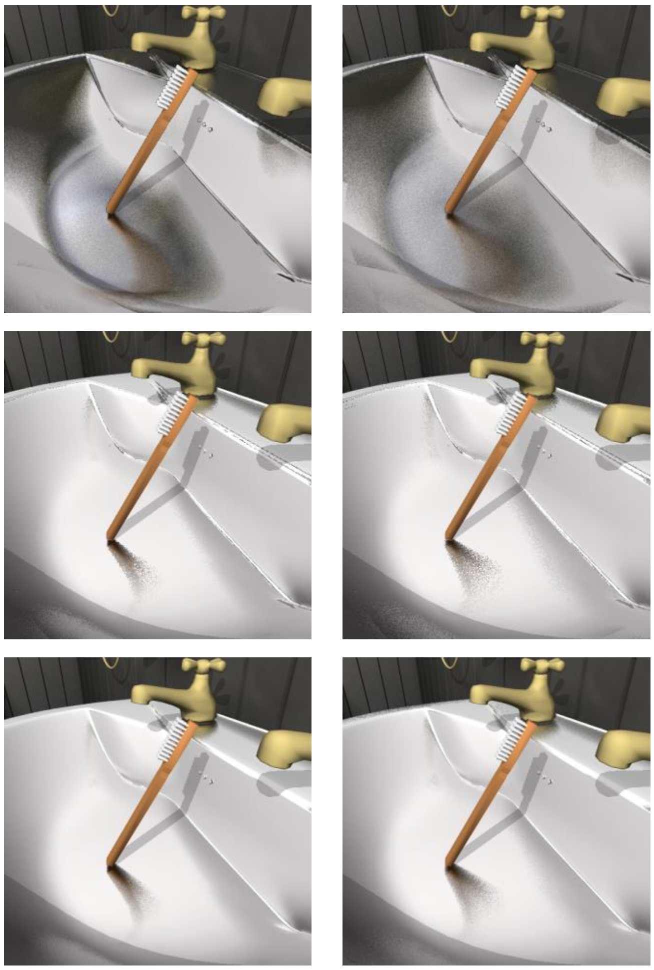
Figure 11. Glossy reflections rendered for two glossiness levels (*left* vs. *right*), with our method and without a diffuse component (*top*), with our method and with a diffuse component (*middle*), and with ray tracing (*bottom*).

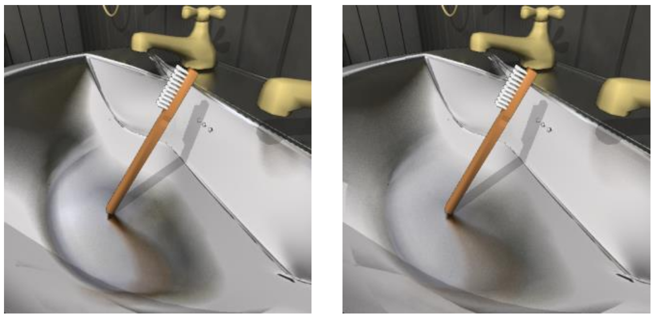
Figure 12. Glossy reflections from Figure 11 now rendered with 49 rays per non-diffuse pixel.

Glossy reflections

Our approach approximates glossy reflections by intersecting multiple reflected rays with a cluster depth image for each cluster pixel. Figure 11 shows that our method achieves quality glossy reflections, comparable to those obtained by ray tracing. For Figure 11 our method uses 8 reflected rays per glossy pixel. Figure 12 shows the less noisy but more expensive glossy reflections obtained with our method when 49 rays per pixel are used.

4.2 Performance

All performance numbers reported in this paper were recorded on a PC workstation with a 3.4GHz Intel(R) Core(TM) i7-2600 CPU, with 4 GB of memory, and with an NVIDIA GeForce GTX 570 graphics card.

Specular (mirror like) reflections

The performance of our method and the comparison to ray tracing is given in Table 1. Ray tracing used the bounding volume hierarchy acceleration scheme (BVH) and the BVH data structure is reconstructed every frame as needed for the dynamic scenes. Performance was measured on two paths of 900 frames each, through each of the two scenes. The scenes contain many reflective surfaces and approximately half the pixels in a frame are non-diffuse. Our method sustains 10fps for all four paths, and it is about 5 times faster than ray tracing for the simpler bathroom scene, and about 10 times faster than ray tracing for the more complex living room scene. To explain the better scalability of our method with scene complexity we breakdown the performance analysis as follows.

Table 1. Performance of our method compared to Optix.

	Non-diffuse pixels [%]		**Optix [fps]**			**Our method [fps]**		
Scene	**max**	**avg**	**max**	**min**	**avg**	**max**	**min**	**avg**
Living room	62	50	1.1	1.0	1.1	25	13	18
	69	58	1.1	0.91	1.0	20	11	15
Bath-room	55	45	3.4	3.1	3.3	20	14	17
	58	44	3.4	3.1	2.3	29	15	18

Table 2 gives the maximum and average times in milliseconds for the main steps of our algorithm (see Section 3.1). As expected, the first pass, which entails rendering the scene geometry with simple shading, and the second pass, which entails binning pixels with a simple pass over the image, take negligible time. The construction of the cluster cameras is expensive, as it requires multiple passes over the pixels in the cluster and concurrent writes for the computation of the cluster point, normal, and AABB. Rendering the cluster depth images is also laborious as it implies a pass over the scene geometry for each cluster. Finally, computing the intersection between the reflected rays and the cluster depth images takes about half the time compared to each of the previous two steps.

Table 2. Performance in milliseconds for various algorithm steps as defined in Section 3.1.

Scene	**First pass & Clustering** *Steps 1 &2*		**Camera construction** *Step 3.a*		**Depth image rendering** *Step 3.b*		**R-DI intersections** *Step 3.c.i*	
	max	**avg**	**Max**	**avg**	**Max**	**avg**	**max**	**avg**
Liv. room	0.9	0.6	30	23	37	26	15	7
	0.8	0.5	28	24	38	28	28	14
Bath-room	0.8	0.5	24	20	39	28	18	11
	0.7	0.5	26	20	30	24	16	8

The rendering parameters that could affect performance are the number of clusters (Table 3), the output frame resolution (Table 4), the cluster depth image resolution (Table 5), and the number of diffuse triangles (Table 6). Tables 3-6 report average performance for the living room scene.

Our method renders the non-diffuse triangles and partitions the resulting non-diffuse pixels into clusters. Consequently, performance does not depend on the number of non-diffuse triangles, but only on the number of clusters (Table 3). All steps depend on the output image resolution: for a bigger output image resolution, the first pass renders a bigger image, there are more pixels to cluster, camera construction handles clusters with more pixels, and the resolution of the depth image for each cluster is higher which translates to longer depth image rendering and reflected ray / depth image intersection times. This translates to slower frame rates for higher resolution (Table 4). However, the cost is not proportional to the number of output image pixels—higher output image resolutions do not increase the complexity of the non-diffuse geometry, thus the number of clusters remains the same, each cluster grouping a larger number of pixels.

Table 3. Average frame rate for various numbers of clusters.

Clusters	**156**	**141**	**123**	**86**	**32**
Frame rate [fps]	12	13	15	15	20

Table 4. Average frame rate for various output resolutions.

Output resolution	**2,048 x 2,048**	**1,024 x 1,024**	**512 x 512**	**256 x 256**
Frame rate [fps]	7.5	15	23	29

Table 5. Average frame rate for various depth image resol's.

Depth image resolution	**1/2 x 1/2**	**1 x 1**	**2 x 2**
Frame rate [fps]	15	15	14

Table 6. Avg. frame rate for various numbers of diffuse tris.

Diffuse triangles [x1,000]	**289**	**162**	**132**	**63**
Frame rate [fps]	10	12	15	15

The frame rate depends very little on the resolution of the depth image—forcing the resolution to be half or twice as much as the resolution computed by the algorithm does not change the frame rate substantially (Table 5). For the steps in Table 2, only the last two depend on the depth image resolution. Step 3.b depends on the number of passes, i.e. the number of depth images, and it depends little on the resolution of individual depth images. Step 3.c.i requires substantially less time than Step 3.b, thus the variable number of steps along the projection of the reflected ray onto the depth image (Section 3.4) has little influence on the overall frame rate. The number of diffuse triangles only affects Step 3.b, and reducing the number of diffuse triangles benefits overall performance (Table 6) until the cost of Step 3.b is too small compared to that of Step 3.a.

We can now explain why our method scales better with scene complexity than ray tracing. For our method, clustering, cluster camera construction, and reflected ray depth image intersection mainly depend on output image resolution. Because of this, simple scenes (e.g. the Cornell box) rendered at high resolution will be handled faster by ray tracing. Whereas depth image construction does depend on scene complexity, the two scenes are still simple enough for depth image construction to take roughly the same amount of time (Table 6). This is not the case for BVH construction needed by ray tracing, which takes substantially longer for the living room scene (910ms) compared to the bathroom scene (260ms). Our method depends on the number of clusters (Table 3), which is similar for all four paths used for Table 1, as shown in Table 7. In conclusion, our method scales better with diffuse scene complexity than ray tracing because the conventional GPU rendering of the cluster depth images scales better than the construction of the acceleration structure needed for ray tracing.

Regarding the dependence on reflector complexity, our method handles optimally planar reflectors, which are frequently encountered in man-made scenes, and ray tracing does not. Complex reflectors, with high curvature and high fragmentation are challenging for both our method and ray tracing: they imply a large number of clusters for our method, and ray tracing has to filter the reflected geometry by shooting tens or hundreds of rays per pixel. Our method filters geometry at a much lower cost during cluster depth image rendering. Both our method and ray tracing are intended for large reflectors that produce coherent reflections; environment mapping should remain the approach of choice for very high complexity reflectors, where it is difficult to judge reflection accuracy and hence the additional cost is not justified.

Table 7. Number of clusters for the four paths in Table 1.

	Living room		Bathroom	
Avg. no. of clusters	148	120	123	107
Max. no. of clusters	195	162	194	158

Glossy reflections

The difference between rendering perfectly specular, mirror-like reflections and rendering glossy reflections consists of intersecting the cluster depth image with multiple reflected rays for each glossy pixel. However, the performance implication is that, in addition to having to intersect multiple rays, it is also the case that each intersection is more expensive. Whereas the pinhole camera constructed for a cluster approximates the one-per-pixel specularly reflected rays well, glossy reflected rays are markedly divergent from the cluster camera rays. Because of this, the length of the segment where a glossy ray projects onto the depth image is longer, leading to more steps for finding the intersection. The glossier the surface (i.e. the more matte and the less mirror-like), the more divergent the rays, and the higher the ray / depth image intersection cost. Glossy rendering performance is given in Table 8. The frame rate is lower compared to that for mirror-like reflections, but the advantage over ray tracing is maintained. Higher performance more approximate glossy reflections can be obtained by intersecting a single ray with the cluster depth image and averaging samples in a neighborhood centered at the intersection.

Table 8. Performance for glossy reflections in Figure 11.

Glossiness level	Rays per pixel	Intersection time [ms]	Frame rate [fps]	
			Optix	Our method
More specular	8	112	1	6
	49	651	0.4	1.4
More matte	8	163	1	5
	49	814	0.4	1.1

In terms of memory requirement, the algorithm scales well since the cluster depth images do not have to be stored in memory simultaneously—the memory is reused as soon as the reflection for a cluster is completed. The total amount of GPU memory required for rendering the 512x512 reflections shown here is 87MB, most of which (i.e. 64MB) is used for the framebuffer with position, normal, cluster ID, and color channels.

4.3 Implementation notes

The first rendering pass that finalizes the diffuse pixels and initializes the non-diffuse pixels (step 1 in Section 3.1) is done on the GPU with a straightforward shader. Non-diffuse pixel clustering (step 2) is also done with a GPU shader, since a pixel is assigned to a cluster using only the information at the pixel, and no information from neighboring pixels.

Cluster camera construction (step 3.a) requires concurrent writes (e.g. for the computation of the cluster centroid, normal, and AABB of sample projections) and we perform the step on the GPU in CUDA using shared memory and atomic operations. The number of costly atomic operations is reduced whenever possible by using regular operations to determine a good initial guess. For example, when searching for a maximum element in an array, running the algorithm without atomic operations will return one of the larger elements of the array. This element is then used to initialize the maximum for the rigorous version of the algorithm that employs atomic operations. Since this initial value is only smaller than a few of the elements of the array, the maximum will only be updated a few times, saving most atomic operations that occur if the maximum is initialized with the customary first element.

Depth image rendering (step 3.b), and finalizing the non-diffuse pixel colors which includes reflected ray / depth image intersection (step 3.c) are performed on the GPU. For step 3.b view frustum culling at object level is used, and for step 3.c intersections for planar reflectors are immediate since the projection of the reflected ray is a single point, i.e. the intersection point (see cluster camera construction in Section 3.3). Handling planar reflectors correctly (no disocclusion errors) and efficiently (planar reflectors are detected automatically in the output frame and only the reflection for the visible part of each planar reflector is computed) is an important strength of our method since planar reflectors are frequently encountered in man-made scenes (Table 9).

Table 9. Percentage of non-diffuse pixels that belong to planar reflectors for the four paths in Table 1.

		Living room		Bathroom	
Planar reflector non-diffuse pixels [%]	**Avg**	41	33	17	20
	Max	52	51	33	34

4.4 Limitations

As discussed above, our method resorts to several approximations. First, the reflected rays are approximated by fitting a conventional planar pinhole camera to each cluster. The smaller the cluster, the better the pinhole's ray approximate the actual reflected rays. Since the image captured by the pinhole is *not* used directly to form the reflection, the approximation only has a second order effect on the correctness of the reflection. In other words, the reflection is *not* distorted, and the samples captured by the pinhole are reflected correctly by computing the intersection with the reflected ray. However, the pinhole does not capture all samples captured by the actual reflected rays, which leads to disocclusion errors as discussed.

A second approximation is that the reflected geometry is replaced with a depth image, which introduces an intermediate resampling of both geometry and color. The effects of this approximation are mitigated by increasing the resolution of the depth images. The third approximation consists of looking up into an environment map the reflected rays generated by non-diffuse pixels of small clusters (e.g. silhouette pixels in Figure 9).

Our method is best suited for mirror-like reflections with one ray per pixel, because glossy rays cannot be approximated well by the cluster cameras, and intersecting a glossy ray with the cluster depth image is more expensive.

Our method requires one rendering pass for each cluster to render the cluster's depth image. These passes cannot be avoided as one has to follow the reflected rays to capture the reflected samples. A cluster camera is constructed with the smallest field of view that encompasses the reflected rays of the cluster. The depth images are non-redundant, except for instances when the same part of the scene is reflected more than once, and except for a small overlap at the borders that ensures reflection continuity between neighboring clusters.

Performance scalability with diffuse scene complexity has to be sought along the lines of reducing the cost of these passes. Partitioning scene geometry with a hierarchical subdivision scheme is of course an option, but that is not suitable for dynamic scenes. Another possibility is to improve the clustering scheme, which in its present form emphasizes efficiency at the cost of unnecessarily numerous clusters. K-means clustering based on k-d trees [42] would result in fewer clusters and we will investigate whether that brings a performance gain sufficient to offset the cost of the slower clustering. We will also investigate grouping clusters with cameras whose frusta are disjoint and rendering one compound depth image for each group in a single pass.

Our method achieves second-order feed-forward rendering, with first order rays being the rays leaving the output image eye. This means that our method supports only first-order reflections. When a reflected ray intersects a reflective surface, the ray color is simply set to the diffuse component of the surface. Higher-order reflections also occur in the case of concave reflectors, which can reflect a ray multiple times until the ray escapes the reflector to sample the environment. We handle concave reflector clusters in one of two ways. One way is to handle concave clusters like the convex clusters—reflected rays are intersected with the cluster depth image, ignoring the second intersection with the reflector surface sampled by the cluster. Another way is to detect that a cluster is concave, by testing whether the center of a cluster is behind the image plane of the camera cluster, and then to intersect reflected rays with the cluster itself to detect a possible second intersection. When such a second intersection occurs, the ray color is set to the diffuse component of the cluster sample. None of the two methods provide the accurate second order reflection, but the second method provides a more stable reflection, at the small additional cost.

Finally, our method brings the most benefit close to the specular end of the specular-glossy-diffuse continuum. A narrower reflection cone per reflective surface point results in more coherent reflected rays that are well approximated by a cluster pinhole camera and requires fewer ray / depth image intersections per pixel. When moving towards the diffuse end of the surface reflectance continuum, the ray coherence decreases and the cluster cameras become panoramas with fields of view that encompass the sum of the upper hemispheres of the cluster pixels. Clustering based on normals does not pay off anymore as the reflected rays at a point become indifferent to the point's normal. We have shown that our method achieves good results for specular reflections and that it can also handle high glossiness. Supporting low glossiness or diffuse reflections require a different strategy for approximating the reflected scene geometry.

5 Conclusions and Future Work

We have presented a method for rendering specular and glossy reflections that achieves quality reflections at interactive rates. No pre-computation is required which supports fully dynamic scenes at substantially higher frame rates than a ray tracer that has to reconstruct its acceleration data structure for every frame. Our method readily works with normal-mapped reflectors (Figure 13).

Compared to methods that approximate the projection of reflected vertices such as the explosion map [27], our method has the advantage of better scalability with reflector complexity and of handling multiple reflections at no extra cost. Our method can handle complex reflectors because it does not attempt to provide a constant time solution to the problem of projecting reflected vertices. Instead, the reflected scene vertices are first mapped to a depth image to which reflected rays are then mapped. The reflected ray to depth image mapping implies a search, which is more expensive than the constant time

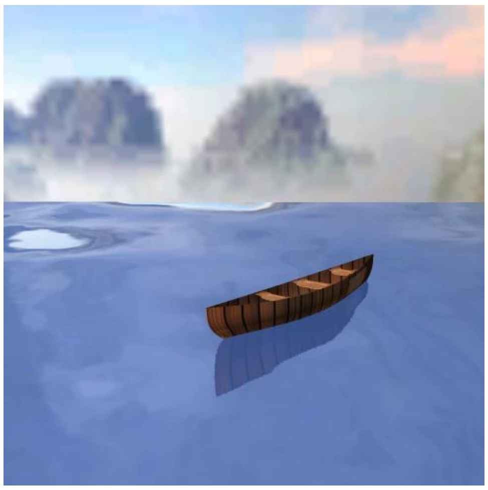
Figure 13. Reflections on a normal-mapped reflector rendered with our method at 31fps.

explosion map projection, but which makes complex reflectors tractable. The search is confined to the 1-D projection of the ray to the depth image, which bounds the cost to the resolution of the depth image.

Compared to image based rendering and caching methods, our method does well for specular reflections which require a high image or cache resolution that makes them expensive to construct and search. Our method compresses the specular reflection data well leveraging the reflected scene geometry. Compared to other methods based on approximating the reflected scene, our method has the advantage of approximating only the part of the scene needed for the reflections in the current frame. The approximation fidelity is tailored to the needs of the current frame. Compared to environment mapping our method is more accurate and compared to methods that rely on view independent approximations of reflected geometry our method is more efficient, enabling applications involving dynamic scenes.

Our method achieves interactive rates on complex reflections. Applications where frame rate is the main design consideration could reserve the use of our method to a subset of the reflective surfaces in a scene. We foresee that the advantage of our method over ray tracing will increase as graphics hardware progresses, since we map well to the GPU’s strength of rendering by projection followed by rasterization.

In addition to the future work directions sketched in Section 4.4, the number of clusters could be reduced by replacing the conventional planar pinhole camera used to approximate the reflected rays of a cluster with more powerful, *non-pinhole* camera models that can conform to larger sets of more diverse rays.

Our method demonstrates that today’s hardware implementation of the feed-forward graphics pipeline is sufficiently versatile and prolific to compute not only the color samples captured by first order rays leaving the eye, but also the samples captured by second order rays. In the context of our paper, the second order rays were created by reflective surfaces, but, in future work, our method could be extended to other types of rays.

6 Acknowledgments

We would like to thank Zheng Yang for help with the implementation and Shuo Wang for help with scene modeling and animation. This work was supported in part by the National Natural Science Foundation of China through Projects 61272349，61190121 and 61190125, by the Macao Science and Technology Development Fund through Project 043/2009/A2, by the National High Technology Research and Development Program of China through 863 Program No. 2013AA01A604, and through Beijing Science Technology Star Plan No. 2009B09.

References

[1] T. Whitted, "An improved illumination model for shaded display", *Communication of the ACM*, vol. 23, no. 6, pp. 343-349, 1980.

[2] A.S. Glassner, "An introduction to ray tracing", *Morgan Kaufmann*, 1989

[3] T. Foley and J. Sugerman, "Kd-tree acceleration structures for a gpu raytracer", *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pp. 15-22, 2005.

[4] N.A. Carr, J. Hoberock, K. Crane and J.C. Hart, "Fast gpu ray tracing of dynmic meshes using geometry images", *Proceedings of Graphics Interface*, pp. 203-209, 2006.

[5] SE. Yoon, C. Lauterbach and D. Manocha, "R-LODs: Fast LOD-Based Ray Tracing of Massive Models", *The Visual Computer*, vol. 22, no. 9-11, pp. 772-784, 2006.

[6] J. Kautz, M.D. McCool, "Approximation of Glossy Reflection with Prefiltered Environment Maps", *Proceedings of Graphics Interface*, pp. 119-126, 2000.

[7] P. Green, J. Kautz and F. Durand, "Efficient reflectance and visibility approximations for environment map rendering", *Computer Graphics Forum*, vol. 26, no. 3, pp. 495-502, 2007.

[8] V. Popescu, E. Sacks and C. Mei, "Sample-Based Cameras for Feed-Forward Reflection Rendering", *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 6, pp. 1590-1600, 2006.

[9] P. Rosen, V. Popescu, K. Hayward and C. Wyman, "Non-Pinhole Approximations for Interactive Rendering", *IEEE Computer Graphics and Applications*, vol. 31, no. 6, pp. 68-83, 2011.

[10] V. Popescu, P. Rosen, L. Arns, X. Tricoche, C. Wyman and C.M. Hoffmann, "The General Pinhole Camera: Effective and Efficient Non-Uniform Sampling for Visualization", *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 5, pp. 777-790, 2010.

[11] I. Wald, P. Slusallek, C. Benthin and M. Wagner, "Interactive rendering with coherent ray tracing", *Computer Graphics Forum*, vol. 20, no. 3, pp.153-165, 2001.

[12] A. Reshetov, A. Soupikov and J. Hurley, "Multi-level ray tracing algorithm", *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 1176-1185, 2005.

[13] T.J. Prucell, I. Buck, W.R. Mark and P. Hanrahan, "Ray tracing on programmable graphics hardware", *ACM Transaction on Graphics*, vol. 21, no. 3, pp. 703-712, 2002.

[14] T.J. Purcell, "Ray tracing on a stream processor", *PhD thesis,* Stanford University, 2004.

[15] Ray tracing on programmable graphics hardware 2005

[16] S. Li, Z. Fan, X. Yin, K. Muller, A.E. Kaufman and X. Gu, "Real-Time Reflection using Ray Tracing with Geometry Field", *Eurographics*, pp. 29-32, 2006.

[17] X. Yu, R. Wang, J. Yu, "Interactive Glossy Reflections using GPU-based Ray Tracing with Adaptive LOD", *Computer Graphics Forum*, vol. 27, no. 7, pp. 1987-1996, 2008.

[18] J. Yu, J. Yang and L. McMillan, "Real-Time Reflection Mapping with Parallax", *Proceeding of Symposium on Interactive 3D graphics and games*, pp. 133-138, 2005.

[19] S.J. Gortler, R. Grzeszczuk, R. Szeliski and M.F. Cohen, "The Lumigraph", *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques 1996*, pp. 43-54, 1996.

[20] M. Levoy and P. Hanrahan, "Light Field Rendering", *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques 1996*, pp.187-196, 1996.

[21] W. Heidrich, H. Lensch, M. F. Cohen and H. Seidel, "Light Field Techniques for Reflections and Refractions", *Proceedings of Eurographics Symposium on Rendering/Eurographics Workshop on Rendering Techniques 1999*, pp.187-196, 1999.

[22] Y. Taguchi, A. K. Agrawal, S. Ramalingam and A. Veeraraghavan, "Axial light field for curved mirrors: Reflect your perspective, widen your view", *Proceedings of Computer Vision and Pattern Recognition 2010*, pp. 499-506, 2010.

[23] J. Kautz and M. D. Mccool, "Approximation of Glossy Reflection with Prefiltered Environment Maps", *Proceedings of Graphics Interface*, pp. 119-126, 2000.

[24] P. Green, J. Kautz and F. Durand, "Efficient Reflectance and Visibility Approximations for Environment Map Rendering", *Computer Graphics Forum*, vol. 26, no. 3, pp. 495--502, 2007.

[25] L. Szirmay-kalos, B. Aszódi, I. Lazányi and M. Premecz, "Approximate Ray-Tracing on the GPU with Distance Impostors", *Computer Graphics Forum*, vol. 24, no. 3, pp. 695--704, 2005.

[26] V. Popescu, C. Mei, J. Dauble and E. Sacks, Reflected-Scene Impostors for Realistic Reflections at Interactive Rates, *Computer Graphics Forum*, vol. 25, no. 3, pp. 313-322, 2006.

[27] E. Ofek and A. Rappoport, "Interactive Reflections on Curved Objects", *Proceedings of the 25th annual conference on Computer graphics and interactive techniques 1998*, pp. 333-342, 1998.

[28] P. Estalella, I. Martín, G. Drettakis and D. Tost , "A GPU-driven Algorithm for Accurate Interactive Reflections on Curved Objects", *Proceedings of the 17th Eurographics conference on Rendering Techniques 2006*, pp. 313-318, 2006.

[29] P.E. Debevec, Y. Yu and G. Borshukov, "Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping", *Proceeding of Eurographics Symposium on Rendering/Eurographics Workshop on Rendering Techniques 1998*, pp. 105-116, 1998.

[30] NVIDIA® OptiX™ ray tracing engine , https://developer.nvidia.com/optix

[31] Leonard McMillan, An Image-Based Approach to Three-Dimensional Computer Graphics, *Ph.D. Dissertation, UNC-CH, Dept. of Computer Science, April 1997.*

[32] Fabio Policarpo and Manuel M. Oliveira., Relief Mapping of Non-Height-Field Surface Details, *ACM SIGGRAPH 2006 Symposium on Interactive 3D Graphics and GamesRedwood City, CA, March 14-17, 2006, pp. 55-62. (ISBN 1-59593-295-X) [DOI].*

[33] Jonathan Shade, Steven Gortler, Li-wei He, and Richard Szeliski, Layered Depth Images, *ACM SIGGRAPH 1998 Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pp. 231-242, 1998.

[34] Kai Bürger, Stefan Hertel, Jens Krüger and Rüdiger Westermann, GPU rendering of secondary effects. *Proceedings of the Vision, Modeling, and Visualization Conference 2007, Saarbrücken, Germany, November 7-9, 2007*, pp. 51-60, 2007.

[35] Matthias Nießner , Henry Schäfer , Marc Stamminger, Fast indirect illumination using layered depth images, *The Visual Computer*, vol. 26 Issue 6-8, pp. 679-686, 2010.

[36] Voicu Popescu, Paul Rosen, and Nicoletta Adamo-Villani, The graph camera, *Transactions on Graphics, Proceedings of Siggraph Asia 2009*, vol. 28, issue 5, 2009.

[37] Gregory J. Ward, Francis M. Rubinstein, Robert D. Clear, A ray tracing solution for diffuse interreflection, *SIGGRAPH 1988 Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pp. 85-92 ,1988.

[38] Gene Greger, Peter Shirley, Philip M. Hubbard, Donald P. Greenberg, The irradiance volume. *IEEE Computer Graphics and Applications*, vol. 27, Issue 2, pp.32-43, 1998.

[39] Gregory J. Ward, Gregory J. Ward, Irradiance gradients. *ACM SIGGRAPH 2008 classes, Article No. 72,* 2008

[40] Jaroslav Kriva´nek, Pascal Gautron, Radiance caching for efficient global illumination. *IEEE Transactions on Visualization and Computer Graphics*, vol. 11, no.5, pp.550-561 2005.

[41] Daniel Scherzer, Chuong H. Nguyen, Tobias Ritschel, Hans-Peter Seidel, Pre-convolved radiance caching. *Computer Graphics Forum*, vol. 31, no. 4, pp. 1391-1397, 2012

[42] Rui Wang, Rui Wang, Kun Zhou, Minghao Pan, Hujun Bao, An Efficient GPU-based Approach for Interactive Global Illumination, *ACM Transactions on Graphics, Proceedings of ACM SIGGRAPH 2009*, vol. 28, no.9, Article No. 91, 2009

[43] Baoquan Liu, Li-Yi Wei, Xu Yang, Ying-Qing Xu, and Baining Guo, Nonlinear beam tracing on a GPU, *Microsoft Technique Report*, 2009.

Lili Wang received the Ph.D. degree from the Beihang University, Beijing, China. She is an associate professor with the School of Computer Science and Engineering of Beihang University, and a researcher with the State Key Laboratory of Virtual Reality Technology and Systems. Her interests include real-time rendering, realistic rendering, global illumination, soft shadow and texture synthesis. （wanglily@buaa.edu.cn）

Naiwen Xie received his B.E. degree in computer science from in Huazhong University of Science and Technology in 2011. He is currently working toward his Ph.D. degree in the State Key Laboratory of Virtual Reality Technology and Systems at Beihang University. His research interests include global illumination, image-based rendering, and image processing.

Wei Ke is a researcher of Macao Polytechnic Institute. He received his Ph.D. from School of Computer Science and Engineering, Beihang University. His research interests include programming languages, functional programming, formal methods and tool support for object-oriented and component-based engineering and systems. His recent research focuses on the design and implementation of open platforms for virtual reality applications, including programming tools, environments and frameworks.

Voicu Popescu received a B.S. degree in computer science from the Technical University of Cluj-Napoca, Romania in 1995, and a Ph.D. degree in computer science from the University of North Carolina at Chapel Hill, USA in 2001. He is an associate professor with the Computer Science Department of Purdue University. His research interests lie in the areas of computer graphics, computer vision, and visualization. His current projects include camera model design, visibility, augmented reality for surgery telementoring, and the use of computer graphics to advance education.